Parallel Programming, CMP3752 - Assignment 1
Executive Summary

Marcus Valerio
25150223

1 Program Overview

The solution developed for this project was altered and developed from the "OpenCL Tutorials" project page on GitHub developed by James Wingate [1]. The first section of the C++ file "Assignment1.cpp" declares string and integer variables which will be initialised later in the code, such as the identification of the platform and device, image of choice and number of bins. Output statements to the command line are then printed, asking the user which image they would prefer to be processed by the program for image enhancement. There are four choices for the user to choose from: a large or a regular sized grayscale image, a large or a regular sized colour image; the colour images were taken from "Tutorial 2" of the "OpenCL Tutorials" GitHub[1]. The user is asked to enter their preference, and while no selection has been made and then the input is in the correct format - an integer from one to four with no whitespace - the user will be asked to enter their preferred number of bins for the Histogram, Cumulative Histogram and Lookup Table. This 'bin request' command line input is processed in the same way as image selection.

A file called "my_kernels.cl" is an OpenCL file which stores the kernels to be used by the C++ code for creating the Histograms essential for enhancement of the image. The data produced and used by these kernels has to be stored in buffers for them to be read from, written to and processed in parallel. The buffers are declared after an OpenCL context and queue, where commands for the device will be pushed, is created. After the memory allocation stage, the kernels are set and executed with the data held in the required buffer as arguments for the kernel's parameters. However, each operation has to be added to the queue before it can be executed on. This process occurs for each kernel and writes data to the buffers which will be outputted to the user. The final section outputs all essential information to the user, such as the input, output, Histogram in a vector form, Cumulative Histogram and Lookup Table. With each of these operations, the total execution time and memory transfer is outputted beneath. The results of producing the final output image from the final kernel can be seen in figure 1.

```
Output processing kernel execution time: 9312 [ns]
Queued 2272, Submitted 9984, Executed 9312, Total 21568 [ns]
```

Fig. 1 - Execution time and memory transfer of producing the enhanced version of "test.pgm" from the "e_output" kernel.

The final output to the user declares the total runtime of the whole program in nanoseconds, which is done by retrieving the profiling information of the first event's queue timestamp and the final event's ending timestamp. The function to get this detail is ".GetFullProfilingInfo" or ".getProfilingInfo" [2] which returns an unsigned 64-bit value in nanoseconds.

A demonstration of the calculation steps for the enhancement algorithm on the large grey scale image, "testLarge.pgm", with 256 bins can be seen in figure 2.

Fig 2.1 - Input image "testLarge.pgm" before enhancement.

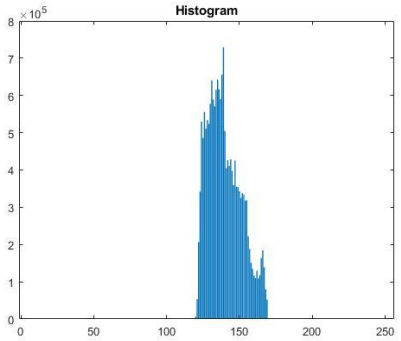

Fig. 2.2 - Histogram of the frequency of intensity levels of the grey scale image.

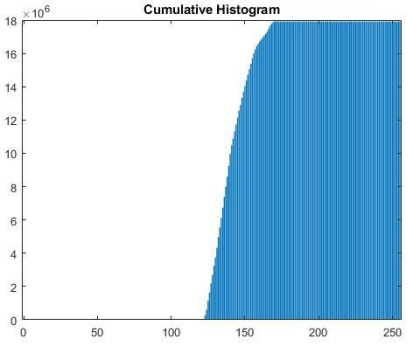

Fig. 2.3 - Cumulative Histogram showing the cumulative cases over the range of the image.

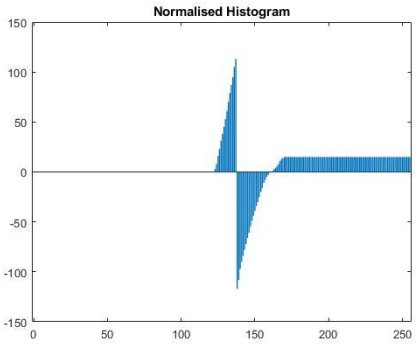

Fig. 2.4 - The Lookup Table normalises the values from the Cumulative histogram, making bright areas brighter and dark areas darker.

Fig. 2.5 - Final output of the enhanced image, “testLarge.pgm”.

2 Results and Analysis

Figure 3 shows how the execution time can vary between different runs of the program, and an average of five runs at least is needed in order to compare the kernel execution times; it was executed on "testLarge.pgm" - which is the large grey scale image - with 256 bins.

Kernel	Run 1 (ns)	Run 2 (ns)	Run 3 (ns)	Run 4 (ns)	Run 5 (ns)	Average (ns)
Histogram	6023168	5940224	5477376	5986304	5991424	5883699
Cumulative Histogram	115712	116736	117760	115712	116736	116531
Lookup Table	7168	6144	7168	6112	8096	6937
Produce Output	149504	150272	153600	149504	149504	150476

Fig. 3 - Table with run times of "testLarge.pgm" with 256 bins.

It can be concluded from figure 3 that producing the normalised histogram (Lookup Table) is one hundred times less time consuming compared to the other tasks in the algorithm. This is because lookup tables are simple estimations based on the data from the cumulative histogram produced before it. Referring to figure 1, we can see that the whole kernel execution is larger with memory transfer as the use of profiling to track the timestamps and memory transfer can incur overhead, which is a source of performance loss due to the extra effort for communication between processors [3].

In figure 4, all four images' average execution times can be compared with 256 bins across the kernels. The large grey scale image takes the most time to execute across all four kernels, especially producing the histogram as it uses atomic increment which increments the bin in a thread-safe manner which leads to performance issues for large datasets due to the serialisation of memory access. However, producing the Cumulative Histogram takes far less time than any other kernel operation as the use of the atomic operation "atomic_add" prevents race conditions, decreasing the number of threads accessing the same resource, thus reducing execution times.

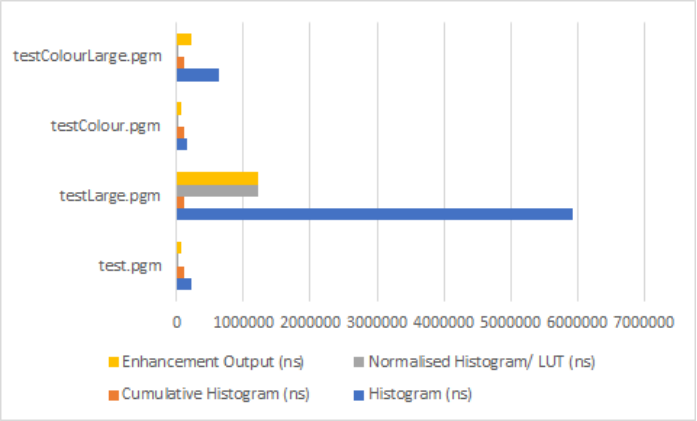

Fig. 4 - Comparison of all four possible images with 526 bins over the whole enhancement algorithm.

The number of bins used for the histograms in the algorithm also affects performance, as seen in figure 5. Using the average execution times, computing with half the number of bins, 128, takes longer to execute than with 256 bins when creating the histogram and the enhanced output image. This is because "[...] you would have less work items or threads trying to access the same bin" [4] when using more bins, as the chance of race conditions is greatly reduced.

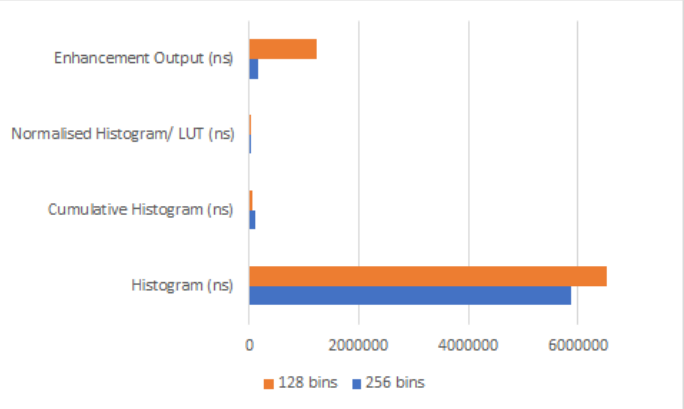

Fig. 5 - Average execution times of the same image, "testLarge.pgm", with 256 bins and 128 bins.

References (4)

[1] - Wingate, J. (2023). *OpenCL Tutorials*. [online] GitHub. Available at: https://github.com/wing8/OpenCL-Tutorials [Accessed 3 Feb. 2023].

[2] - man.opencl.org. (n.d.). *clGetEventProfilingInfo*. [online] Available at: https://man.opencl.org/clGetEventProfilingInfo.html [Accessed 10 Mar. 2023].

[3] - Ghafoor, M. (2023). *Lecture 8: Performance & optimisation.* [online] Available at: https://blackboard.lincoln.ac.uk/ultra/courses/_181985_1/cl/outline [Accessed 29 Mar. 2023].

[4] - Ghafoor, M. (2023). *Lecture 7: Histogram - Parallelisation.* [online] Available at: https://blackboard.lincoln.ac.uk/ultra/courses/_181985_1/cl/outline [Accessed 22 Mar. 2023].